COMP1511 - Programming Fundamentals

Week 3 - Lecture 6

What did we learn last lecture?

Code Style and Code Reviews

- Coding for and with humans

Introduction of Functions

- Separating code for reuse

What are we covering today?

Computers as theoretical tools

- Fundamentals of what a computer is
- How we use memory in C

Arrays

- Using multiple variables at once

What is a computer?

At the most fundamental level . . .

- A processor that executes instructions
- Some memory that holds information

The Turing Machine

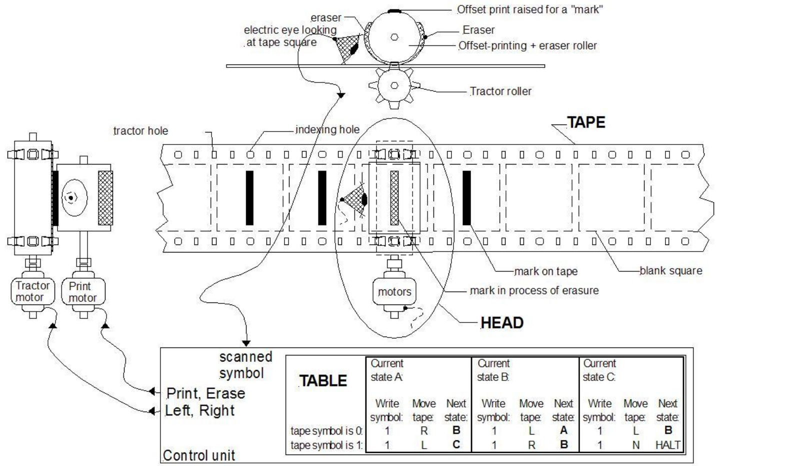
Originally a theoretical idea of computation

- There is a tape that can be infinitely long
- We have a “head” that can read or write to this tape
- We can move the head along to any part of the tape
- There’s a “state” in which the machine remembers its current status
- There’s a set of instructions that say what to do in each state

Turing Machines

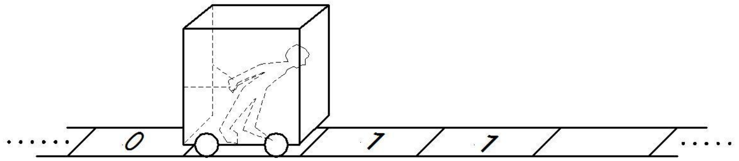
Some images of Turing Machines

- A tape and a read/write head
- Some idea of control of the head



	Current state A:			Current state B:			Current state C:		
	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:
tape symbol is 0:	1	R	**B**	1	L	**A**	1	L	**B**
tape symbol is 1:	1	L	**C**	1	R	**B**	1	N	HALT

A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.



Images from Wikipedia (https://en.wikipedia.org/wiki/Turing_machine_gallery)

The Processor

We also call them Central Processing Units (CPUs)

- Maintains a “state”
- Works based on a current set of instructions
- Can read and write from/to memory

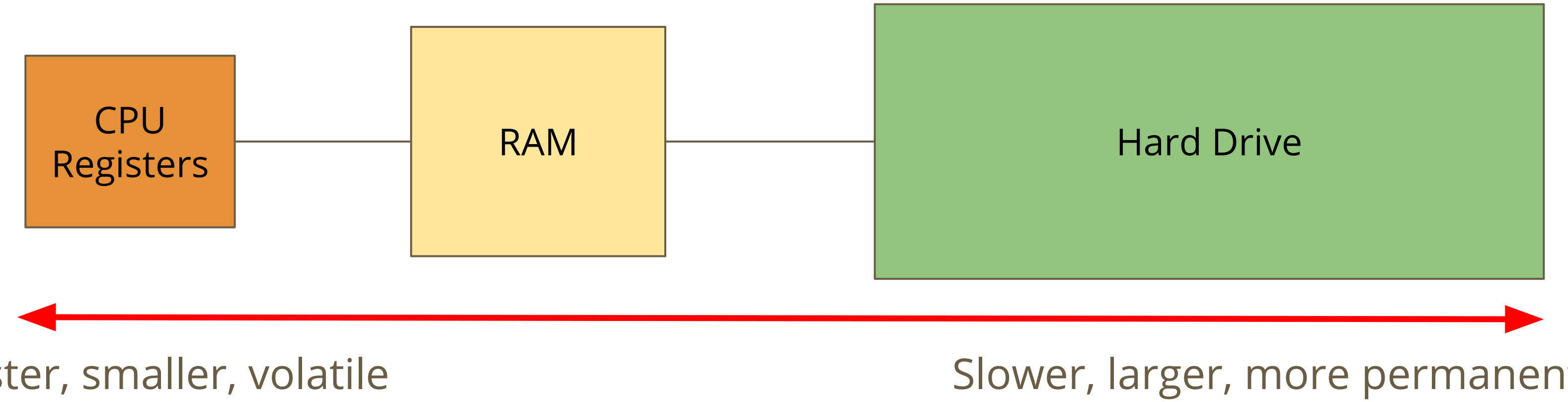
In our C Programming

- State - where are we up to in the code right now
- Instructions - compiled from our lines of code
- Reading/Writing - Variables

Memory

All forms of Data Storage on a computer

- From registers (tiny bits of memory on the CPU) through Random Access Memory (RAM) and to the Hard Disk Drive. All of these are used to store information



How does C use memory

- On the **Hard Drive**
- Our C source code files are stored on our Hard Drive
- Dcc compiles our source into another file, the executable program

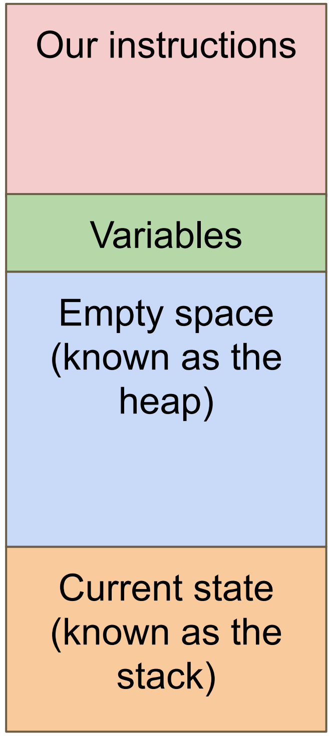
- In **Random Access Memory**
- When we run our program, all the instructions are copied into RAM
- Our CPU will work through memory executing our instructions in order
- Our variables are stored in RAM as well
- Reading and writing to variables will change the numbers in RAM

A snapshot of a program in memory

What happens in memory when we run a program?

- Our Operating System gives us a chunk of memory
- Our program copies its instructions there

- Some space is reserved for declared variables
- The **Stack** is used to track the current state
- The stack grows and shrinks as the program runs
- The **Heap** is empty and ready for use
- We can use the heap to store data while the program is running


Our instructions
Variables
Empty space (known as the heap)
Current state (known as the stack)

There's more … later

Computers and programs are highly complex

- This was just an overview
- As you go through your learning, you will unlock more information
- For now, we have enough understanding to continue using C

Arrays

When we need a collection of variables together

- Sometimes we need a bunch of variables of the same type
- We also might need to process them all
- Our current use of ints and doubles might not be able to handle this

Let’s take a look at our current capability (and why we need arrays) . . .

An Example

Let’s record everyone’s marks at the end of the term

- We could do this as a large collection of integers . . .

```
int main (void) {
    int marksStudent1;
    int marksStudent2;
    int marksStudent3;
    int marksStudent4;
    // etc

```

If we want to test all these ints

We'd need a whole bunch of nearly identical if statements

In this situation

- There's no way to loop through the integers
- Having to rewrite the same code is annoying and hard to read or edit
- So let's find a better way . . .

```
int main (void) {
    int marksStudent1;
    int marksStudent2;
    int marksStudent3;
    int marksStudent4;
    // etc

    if (marksStudent1 >= 50) {
        // pass
    }
    if (marksStudent2 >= 50) {
        // pass
    }
    // etc
```

An Array of Integers

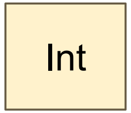
If our integers are listed as a collection

- We’ll be able to access them as a group
- We’ll be able to loop through and access each individual element

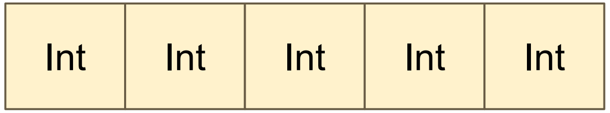
Arrays

What is an array?

- A variable is a small amount of memory
- An array is a larger amount of memory that contains multiple variables
- All of the elements (individual variables) in an array are the same type
- Individual elements don't get names, they are accessed by an integer index



A single integer worth of memory



An array that holds 5 integers

Declaring an Array

Similar, but more complex than declaring a variable

```
int main (void) {
    // declare an array
    int arrayOfMarks[10] = {0};

```

- `int` - the type of the variables stored in the array
- `[10]` - the number of elements in the array
- `= {0}` - Initialises the array as all zeroes
- `= {0,1,2,3,4,5,6,7,8,9}` - Initialises the array with these values

Array Elements

- An element is a single variable inside the array
- They are accessed by their index, an int that is like their address
- Indexes start from 0
- Trying to access an index outside of the array will cause errors

	0	1	2	3	4	5	6	7	8	9
arrayOfMarks	55	70	44	91	82	64	62	68	32	72

In this example, element 2 of arrayOfMarks is 44 and element 6 is 62

Accessing elements in C

C code for reading and writing to individual elements

```
int main (void) {
    // declare an array, all zeroes
    int arrayOfMarks[10] = {0};

    // make first element 85
    arrayOfMarks[0] = 85;
    // access using a variable
    int accessIndex = 3;
    arrayOfMarks[accessIndex] = 50;
    // copy one element over another
    arrayOfMarks[2] = arrayOfMarks[6];
    // cause an error by trying to access out of bounds
    arrayOfMarks[10] = 99;
```

Reading and Writing

Printf and scanf with arrays

- We can’t printf a whole array
- We also can’t scanf a line of user input text into an array
- We can do it for individual elements though!

The trick then becomes looping to access all individual elements one by one

User input/output with Arrays

Using printf and scanf with Arrays

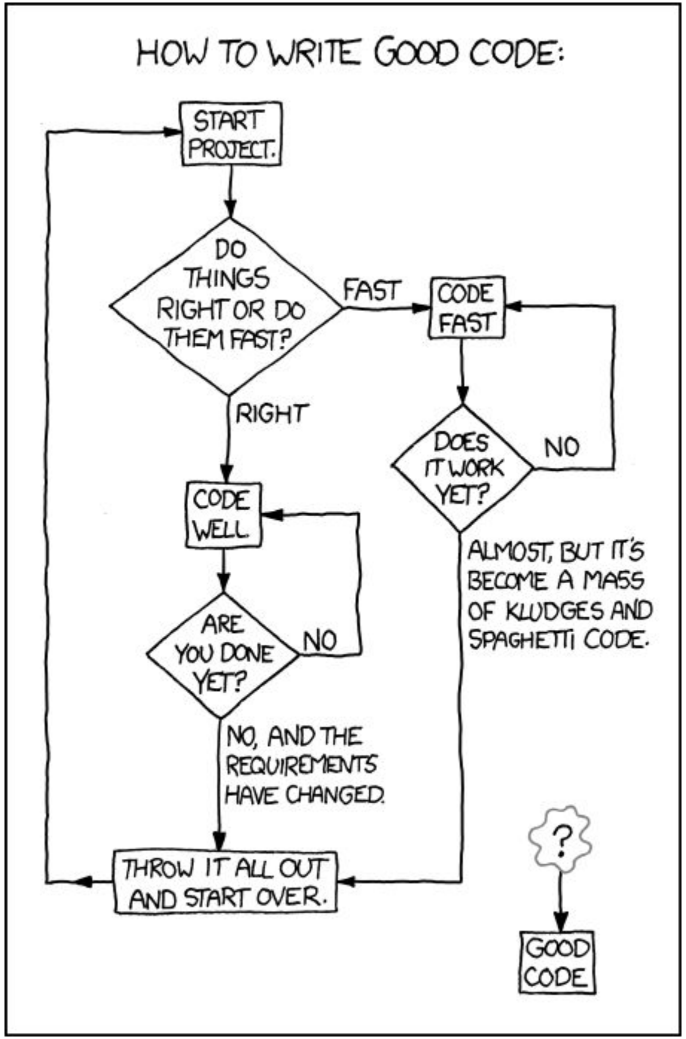
```
int main (void) {
     // declare an array, all zeroes
     int arrayOfMarks[10] = {0};

     // read from user input into 3rd element
     scanf(“%d”, &arrayOfMarks[2]);
     // output value of 5th element
     printf(“The 5th Element is: %d”, arrayOfMarks[4]);

     // the following code DOES NOT WORK
     scanf(“%d %d %d %d %d %d %d %d %d %d”, &arrayOfMarks);
```

Break Time

Writing "good" code

- It's never going to be easy!
- It's an ongoing struggle

https://xkcd.com/844/


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
RIGHT
DOES IT WORK YET?
NO
CODE WELL.
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
?
THROW IT ALL OUT AND START OVER.
GOOD CODE

A Basic Program using Arrays

Let's make a program to track player scores in a game

- We have four players that are playing a game together
- We want to be able to set and display their scores
- We also want to be able to see who's winning and losing the game
- The game needs to know how many points have been scored in total, so we'll also want some way of calculating that total

Break down the program

What are the individual elements we need to make?

- First we create an array
- Then we use indexes to access the individual players and enter scores
- We're going to need while loops to step through the array
- Most of the extra functionality we want will be done by looping through the array

Create the Array and populate it

Setting the elements using indexes (manually for now)

```
#include <stdio.h>

#define NUM_PLAYERS 4

int main(void) {
    int scores[NUM_PLAYERS] = {0};
    int counter;

    // assigning values directly to indexes
    scores[0] = 55;
    scores[1] = 84;
    scores[2] = 32;
    scores[3] = 61;
```

Let’s loop through and see those values

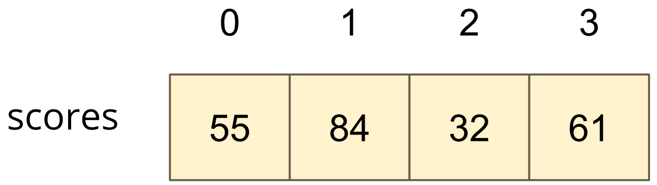
Accessing all array elements by looping

This is a pretty good candidate for code to put in a function!

```
// continued from last slide
// loop through and display all scores
int counter = 0;
while (counter < NUM_PLAYERS) {
    printf(
        "Player %d has scored %d points.\n",
        counter,
        scores[counter]
    );
    counter++;
}
```

Now that we have our array

It will look a bit like this:



Next, we can loop through to find:

- The lowest
- The highest
- And the total

Finding particular values in an array

If we see all the values, we can easily find the highest

- We'll loop through all the values in the array
- We'll save the highest value we've seen so far
- Then replace it if we find something higher
- By the time we reach the end, we will have the highest value

Finding the highest score

We could put this in a separate function also!

```
int highest = 0;
int indexHighest = -1;
counter = 0;
while (counter < NUM_PLAYERS) {
    if (scores[counter] > highest) {
        highest = scores[counter];
        indexHighest = counter;
    }
    counter++;
}
printf(
    "Player %d has the highest score of %d.\n",
    indexHighest, highest
);
```

Finding the Total

This is even easier than the highest!

We just add all the values to a variable we're keeping outside the loop

```
int total = 0;
counter = 0;
while (counter < NUM_PLAYERS) {
    total += scores[counter];
    counter++;
}
printf("Total points scored across the players is %d", total);
```

Wait, what was that new syntax?

+= is another shorthand operator

It's used for accumulating values in a variable

```
int a = 0;
int b = 0;

// These two lines of code will do the same thing
a += 5;
b = b + 5;

// both a and b are now equal to 5
```

What about input into an array

Remember, we can't access the whole array, only individual elements

But we can definitely loop through the array entering values!

```
    // assigning scores using user input
    counter = 0;
    while (counter < NUM_PLAYERS) {
        printf("Please enter Player %d's score: ", counter);
        scanf("%d", &scores[counter]);
        counter++;
    }
```

A Score Tracker

We've built our first program using an array (and maybe some functions)

- We've accessed elements by index to set their values
- We've looped through to access values to output
- We've looped through to find highest and lowest
- We learnt about accumulating values
- We've also looked at reading values into the array
- We've seen how we can separate code into a function

What did we learn today?

Theory of a Computer

- A processor - carries out operations
- Some memory - stores information

Arrays

- Collections of identical variables
- Individual elements are accessed by indexes